Complete Knowledge Assumption

- Often you want to assume that your knowledge is complete. Everything not known to be true is false.

Complete Knowledge Assumption

- Often you want to assume that your knowledge is complete. Everything not known to be true is false.
- Example: you can state what switches are up and the agent can assume that the other switches are down.

Complete Knowledge Assumption

- Often you want to assume that your knowledge is complete. Everything not known to be true is false.
- Example: you can state what switches are up and the agent can assume that the other switches are down.
- Example: assume that a database of what students are enrolled in a course is complete.

Complete Knowledge Assumption

- Often you want to assume that your knowledge is complete. Everything not known to be true is false.
- Example: you can state what switches are up and the agent can assume that the other switches are down.
- Example: assume that a database of what students are enrolled in a course is complete.
- The definite clause language is monotonic: adding clauses can't invalidate a previous conclusion.

Complete Knowledge Assumption

- Often you want to assume that your knowledge is complete. Everything not known to be true is false.
- Example: you can state what switches are up and the agent can assume that the other switches are down.
- Example: assume that a database of what students are enrolled in a course is complete.
- The definite clause language is monotonic: adding clauses can't invalidate a previous conclusion.
- Under the complete knowledge assumption, the system is non-monotonic: adding clauses can invalidate a previous conclusion.

Complete Knowledge Assumption

- Often you want to assume that your knowledge is complete. Everything not known to be true is false.
- Example: you can state what switches are up and the agent can assume that the other switches are down.
- Example: assume that a database of what students are enrolled in a course is complete.
- The definite clause language is monotonic: adding clauses can't invalidate a previous conclusion.
- Under the complete knowledge assumption, the system is non-monotonic: adding clauses can invalidate a previous conclusion.
- The complete knowledge assumption is sometimes called the closed world assumption.

Completion of a knowledge base

- Suppose the rules for atom a are

$$a \leftarrow b_1.$$
$$\vdots$$
$$a \leftarrow b_n.$$

  equivalent logical formula $a \leftarrow b_1 \vee \ldots \vee b_n$.
  "a is true if b_1 or $\ldots$ or b_n"

Completion of a knowledge base

- Suppose the rules for atom a are

$$a \leftarrow b_1.$$
$$\vdots$$
$$a \leftarrow b_n.$$

  equivalent logical formula $a \leftarrow b_1 \vee \ldots \vee b_n$.
  "a is true if b_1 or ... or b_n"

- Under the Complete Knowledge Assumption, if a is true, one of the b_i must be true:

$$a \rightarrow b_1 \vee \ldots \vee b_n.$$

Completion of a knowledge base

- Suppose the rules for atom a are

$$a \leftarrow b_1.$$
$$\vdots$$
$$a \leftarrow b_n.$$

  equivalent logical formula $a \leftarrow b_1 \vee \ldots \vee b_n$.
  "a is true if b_1 or $\ldots$ or b_n"

- Under the Complete Knowledge Assumption, if a is true, one of the b_i must be true:

$$a \rightarrow b_1 \vee \ldots \vee b_n.$$

- Under the CKA, the clauses for a mean Clark's completion:

$$a \leftrightarrow b_1 \vee \ldots \vee b_n$$

  "a is true if and only if b_1 or $\ldots$ or b_n"

Clark's Completion of a KB

- Clark's completion of a knowledge base consists of the completion of every atom.
- An atom h with no clauses, has the completion

Clark's Completion of a KB

- Clark's completion of a knowledge base consists of the completion of every atom.
- An atom h with no clauses, has the completion
  $h \leftrightarrow \mathit{false}$.
  "h is false".

Clark's Completion of a KB

- Clark's completion of a knowledge base consists of the completion of every atom.
- An atom h with no clauses, has the completion
  $h \leftrightarrow false$.
  "h is false".
- You can interpret negations in the body of clauses.

  $$\sim h$$

  means that h is false under the complete knowledge assumption
  This is negation as failure.

Electrical Environment (`elect_naf.pl`)

Idea: only represent `up` and use `\+ up` instead of `down`

- Easier to specify
- Less error prone (exactly one must be true)

Negation as failure example (naf.pl)

$$p \leftarrow q \wedge \sim r.$$

$$p \leftarrow s.$$

$$q \leftarrow \sim s.$$

$$r \leftarrow \sim t.$$

$$t.$$

$$s \leftarrow w.$$

Bottom-up negation as failure interpreter

$C := \{\}$
repeat
  either
    select $r \in KB$ such that
      r is "$h \leftarrow b_1 \wedge \ldots \wedge b_m$"
      $b_i \in C$ for all i, and
      $h \notin C$
    $C := C \cup \{h\}$

Bottom-up negation as failure interpreter

$C := \{\}$
repeat
    either
        select $r \in KB$ such that
            r is "$h \leftarrow b_1 \wedge \ldots \wedge b_m$"
            $b_i \in C$ for all i, and
            $h \notin C$
        $C := C \cup \{h\}$
    or
        select h such that for every rule "$h \leftarrow b_1 \wedge \ldots \wedge b_m$" $\in KB$
                either for some $b_i, \sim b_i \in C$
                or some $b_i = \sim g$ and $g \in C$
        $C := C \cup \{\sim h\}$
until no more selections are possible

Negation as failure example

$$p \leftarrow q \wedge \sim r.$$

$$p \leftarrow s.$$

$$q \leftarrow \sim s.$$

$$r \leftarrow \sim t.$$

$$t.$$

$$s \leftarrow w.$$

Top-Down negation as failure proof procedure

- If the proof for a fails, you can conclude $\sim a$.
- Failure can be defined recursively:
  Suppose you have rules for atom a:

$$
\begin{aligned}
a &\leftarrow b_1 \\
&\vdots \\
a &\leftarrow b_n
\end{aligned}
$$

  If each body b_i fails, a fails.

Top-Down negation as failure proof procedure

- If the proof for a fails, you can conclude $\sim a$.
- Failure can be defined recursively:
  Suppose you have rules for atom a:

$$\begin{aligned} a &\leftarrow b_1 \\ &\vdots \\ a &\leftarrow b_n \end{aligned}$$

  If each body b_i fails, a fails.
  A body fails if one of the conjuncts in the body fails.

Top-Down negation as failure proof procedure

- If the proof for a fails, you can conclude $\sim a$.
- Failure can be defined recursively:
  Suppose you have rules for atom a:

$$
\begin{aligned}
a &\leftarrow b_1 \\
&\vdots \\
a &\leftarrow b_n
\end{aligned}
$$

  If each body b_i fails, a fails.
  A body fails if one of the conjuncts in the body fails.
- If there are no rules for h

Top-Down negation as failure proof procedure

- If the proof for a fails, you can conclude $\sim a$.
- Failure can be defined recursively:
  Suppose you have rules for atom a:
  $$\begin{aligned} a &\leftarrow b_1 \\ &\vdots \\ a &\leftarrow b_n \end{aligned}$$
  If each body b_i fails, a fails.
  A body fails if one of the conjuncts in the body fails.
- If there are no rules for h then h fails

Top-Down negation as failure proof procedure

- If the proof for a fails, you can conclude $\sim a$.
- Failure can be defined recursively:
  Suppose you have rules for atom a:

$$a \leftarrow b_1$$
$$\vdots$$
$$a \leftarrow b_n$$

  If each body b_i fails, a fails.
  A body fails if one of the conjuncts in the body fails.
- If there are no rules for h then h fails
- Note that you need *finite* failure. Example $p \leftarrow p$.

Default Reasoning

- When giving information, we don't want to enumerate all of the exceptions, even if we could think of them all.
- In default reasoning, we specify general knowledge and modularly add exceptions. The general knowledge is used for cases we don't know are exceptional.
- Classical logic is monotonic: If g logically follows from A, it also follows from any superset of A.
- Default reasoning is nonmonotonic: When we add that something is exceptional, we can't conclude what we could before.

Example: default reasoning about resorts (beach.pl)

- A resort is on the beach or away from the beach.

Example: default reasoning about resorts (beach.pl)

- A resort is on the beach or away from the beach.
  A resort is away from the beach unless it says it is on a beach.

Example: default reasoning about resorts (beach.pl)

- A resort is on the beach or away from the beach.
  A resort is away from the beach unless it says it is on a beach.

$$away\_from\_beach \leftarrow \sim on\_beach.$$

Example: default reasoning about resorts (beach.pl)

- A resort is on the beach or away from the beach.
  A resort is away from the beach unless it says it is on a beach.

  $$away\_from\_beach \leftarrow \sim on\_beach.$$

- If we are told the resort is on the beach, we would expect that resort users would have access to the beach.
  If they have access to a beach, we would expect them to be able to swim at the beach.

Example: default reasoning about resorts (beach.pl)

- A resort is on the beach or away from the beach.
  A resort is away from the beach unless it says it is on a beach.

  $$away\_from\_beach \leftarrow \sim on\_beach.$$

- If we are told the resort is on the beach, we would expect that resort users would have access to the beach.
  If they have access to a beach, we would expect them to be able to swim at the beach.

  $$beach\_access \leftarrow on\_beach \wedge \sim ab\_beach\_access.$$

Example: default reasoning about resorts (beach.pl)

- A resort is on the beach or away from the beach.
  A resort is away from the beach unless it says it is on a beach.

  $$away\_from\_beach \leftarrow \sim on\_beach.$$

- If we are told the resort is on the beach, we would expect that resort users would have access to the beach.
  If they have access to a beach, we would expect them to be able to swim at the beach.

  $$beach\_access \leftarrow on\_beach \land \sim ab\_beach\_access.$$

  $$swim\_at\_beach \leftarrow beach\_access \land \sim ab\_swim\_at\_beach.$$

Example: default reasoning about resorts (beach.pl)

- A resort is on the beach or away from the beach.
  A resort is away from the beach unless it says it is on a beach.

  $$away\_from\_beach \leftarrow \sim on\_beach.$$

- If we are told the resort is on the beach, we would expect that resort users would have access to the beach.
  If they have access to a beach, we would expect them to be able to swim at the beach.

  $$beach\_access \leftarrow on\_beach \land \sim ab\_beach\_access.$$

  $$swim\_at\_beach \leftarrow beach\_access \land \sim ab\_swim\_at\_beach.$$

  $$ab\_swim\_at\_beach \leftarrow enclosed\_bay \land big\_city \land \sim ab\_no\_swim.$$

  $$ab\_no\_swim \leftarrow in\_BC \land \sim ab\_BC\_beaches.$$

See end of `logicNegation.py` in aipython.org or
`https://artint.info/3e/resources/ch05/beach.pl`

Default Example

How can we represent

- Birds fly.

Default Example

How can we represent

- Birds fly.
- Emus and tiny birds dont fly.

Default Example

How can we represent

- Birds fly.
- Emus and tiny birds dont fly.
- Hummingbirds are exceptional tiny birds.